

Vorlesungsmitschrift

# Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

## Inhaltsverzeichnis

<b>1</b>	<b>Vorlesung 1 - 17.04.</b>	<b>1</b>
1.1	Was sind Programmiersprachen? . . . . .	1
1.2	Mehrdeutigkeit in natürlichen Sprachen . . . . .	1
1.3	Formalisierungsmethoden . . . . .	1
1.4	Referenzsprache . . . . .	2
1.4.1	Definition der Syntax . . . . .	2
<b>2</b>	<b>Vorlesung 2 - 24.04.</b>	<b>4</b>
2.1	Operationelle Semantik am Beispiel der Terme . . . . .	4
2.1.1	Terme der Sprache WHILE . . . . .	4
2.1.2	Beispiel: AST . . . . .	4
2.1.3	Informelle Semantik . . . . .	4

# 1 Vorlesung 1 - 17.04.

## 1.1 Was sind Programmiersprachen?

**Definition 1.1** (Programmiersprachen).

*Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.*

Memo technischer Begriff -> z.b. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

**Syntax** einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)

- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

**Semantik** (diese Vorlesung)

- Bedeutung
- Interpretation

Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

**Fokus:** Formalisierung

**Pragmatik** Fragen nach dem Gebrauch und Zweck (Useability).

*Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)*

## 1.2 Mehrdeutigkeit in natürlichen Sprachen

**Synonyme** *Schloss, Schimmel, ...*

Auflösung durch Kontext (meist leicht und unproblematisch)

**Satzebene** *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*

Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

## 1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

### Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

### Entwicklung der Formalisierungsansätze

**operationale Semantik** (*Landin 1964*): Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnlich wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

**denotationelle Semantik** (*McCarthy 1962*): Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

**axiomatische Semantik** (Hoare 1969): Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel:  $\{Q\}P\{R\}$

## 1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

### 1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)

Elementare Einheiten

---

```

1 // ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
2 Z ::= 0 | 1 | ... | MAX | -1 | -2 | ... | MIN
3 // Wahrheitswerte BOOL
4 W ::= TRUE | FALSE
5 // Konstanten KON
6 K ::= Z | W
7 // Bezeichner bzw. Variablen mit Indizes
8 I ::= a | b | ... | z | a1 | a2 | ... | zi
9 // Operatoren
10 OP ::= + | - | * | / | mod
11 // boolesche Operatoren
12 BOP ::= < | > | = | !> | !< | !=

```

---

Zusätzliche Einheiten (induktiv)

---

```

1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | read, für T1,T2 in TERM
3 // boolesche Terme BT
4 B ::= W | T1 BOP T2 | read | not B
5 // Befehle (Zustandstransformation) COM
6 C ::= skip | I := T | C1; C | if B then C1 else C2 | while B do C |
    output T | output B

```

---

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!

Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

- Typkonflikte
- Fehlerbehandlung
- Rekursion

### WHILE ist mehrdeutig?

Ja, das zeigt folgendes Beispiel:

---

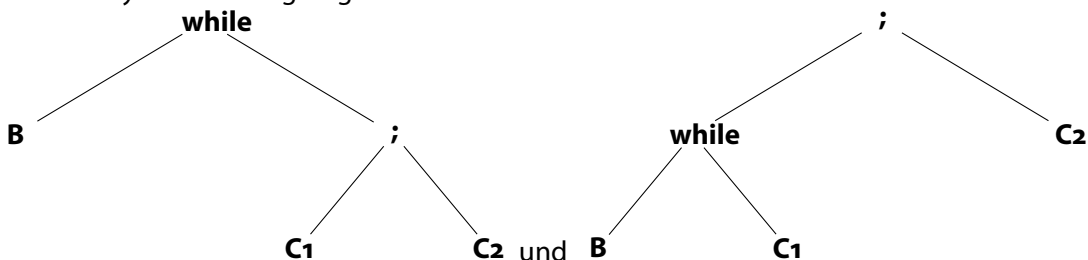
```

1 while B do C1; C2

```

---

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

---

```
1 while B do
2   C1;
3   C2
```

---

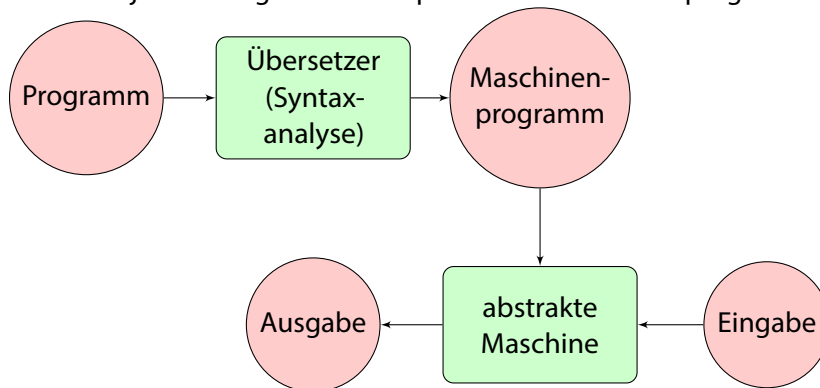
## 2 Vorlesung 2 - 24.04.

### 2.1 Operationelle Semantik am Beispiel der Terme

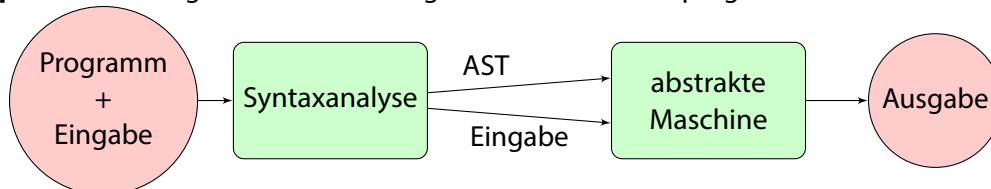
Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen! *(Inhalt ist nicht im Lehrbuch!)*

Grundsätzlich gibt es zwei Methoden:

**Übersetzer** Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



**Interpreter** Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



**AST:** abstract syntax tree

#### 2.1.1 Terme der Sprache WHILE

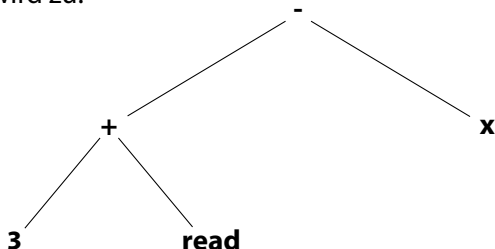
```
1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | read, für T1,T2 in TERM
```

#### 2.1.2 Beispiel: AST

Der Ausdruck

```
1 3 + read - x
```

wird zu:

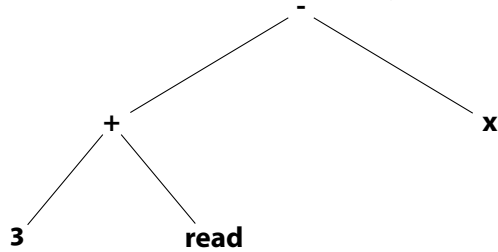


#### 2.1.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.

**Annahme:** wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

**Übersetzer Idee:** depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

---

```

1 PUSH 3
2 READ
3 ADD
4 LOAD x
5 SUB
  
```

---

Zustandsveränderungen:  
aktueller Zustand (siehe Bild Architektur!):

$\langle \epsilon   S   8.5 \dots \rangle$	$\xrightarrow{\text{PUSH } 3}$	$\langle 3   S   8.5 \dots \rangle$
	$\xrightarrow{\text{READ}}$	$\langle -8.3 - \epsilon   S   5 \dots \rangle$
	$\xrightarrow{\text{ADD}}$	$\langle 3 + (-8) \cdot \epsilon   S   5 \dots \rangle$
	$\xrightarrow{\text{LOAD } x}$	$\langle 2 \cdot -5 \cdot \epsilon   S   \dots 5 \rangle$
	$\xrightarrow{\text{SUB}}$	$\langle -7 \cdot \epsilon   S   5 \dots \rangle$

Semantik eines Terms  $T$  zu geg. Speicher  $S$  und Eingabe  $E$  ist die Spitze des Wertekellers (STACK) nach Ausführung von  $\text{trans } T$  auf  $\langle \epsilon | S | E \rangle$ , falls diese Ausführung fehlerfrei läuft, sonst Fehler!

**Interpreter** (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
  - \* Wertekeller  $W (\in Z\text{AHL}^*)$
  - \* Speicher  $S (S \in [ID \rightarrow Z\text{AHL}])$
  - \* Kontrollkeller  $K (\in (AST \cup OP)^*)$
  - \* Eingabe  $E (\in Z\text{AHL}^*)$

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum  $Z$  durch Angabe von:

- (i) einem Anfangszustand  $Z_{T,S,E}$  für jeden Term  $T$ , Speicher  $S$  und Eingabe  $E$ .
- (ii) eine Zustandsüberföhrungsfunktion  $\Delta : Z \rightarrow Z$  (partiell)
- (iii) Erklärung der Semantik über Iteration von  $\Delta$

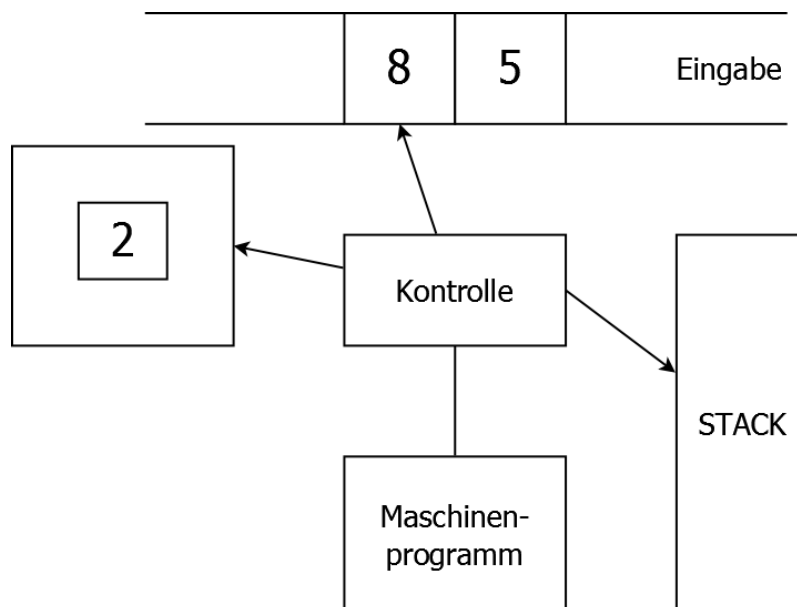
für Terme aus WHILE:

- (i)  $Z_{T_0, S_0, E_0} := \langle \epsilon | S_0 | T_0 \cdot \epsilon | E_0 \rangle$
- (ii)  $\Delta$  per Induktion über die Struktur der Kontrollkellerspitze

$$\begin{aligned}
\Delta \langle W|S|n.K|E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in ZAHL, W, S, E \text{ wie oben.} \\
\Delta \langle W|S|x.K|E \rangle &= \langle s(x).W|S|K|E \rangle \text{ für alle } x \in ID \\
\Delta \langle W|S|read.K|n.E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in ZAHL \\
\Delta \langle W|S|T_1OPT_2.K|E \rangle &= \langle W|S|T_1.T_2.OP.K|E \rangle \\
\Delta \langle n_2.n_1.W|S|OP.K|E \rangle &= \langle n_1OPn_2.W|S|K|E \rangle \text{ falls } n_1, n_2 \in ZAHL \text{ falls } n_1OPn_2 \text{ definiert ist.}
\end{aligned}$$

- (iii) Die Semantik eines (beliebigen) Terms  $T$  im Bezug auf einem Speicher  $S$  und eine Eingabe  $E$  ist  $n \in ZAHL$ , wenn  $\Delta^k Z_{T,S,E} = \langle n.\epsilon|S|\epsilon|E' \rangle$  für beliebige  $E' \in ZAHL^*$ , undefiniert sonst!

### Architektur der abst. Maschine



### Befehlssatz

---

```

1 // Stack und Speicher Operationen
2 READ      // lese Zahl v. Eingabe, PUSH Zahl, rücke Zeiger um eins weiter
3 LOAD x    // PUSH Inhalt aus Speicher mit symbol. Adr. x
4 PUSH n    // für jede Zahl aus N lege n auf Stack
5 (STORE x) // belegt Speicher mit der symbolischer Adresse x
6 (GOTO n)  // bedingter Sprung
7
8 // arithmetische Operationen
9 ADD
10 MULT
11 SUB
  
```

---